Software Reliability Modelling and Prediction with Hidden Markov Chains

Jean-Baptiste Durand
INRIA Rhône-Alpes
Grenoble, France
Jean-Baptiste.Durand@cirad.fr

Olivier Gaudoin
Institut National Polytechnique de Grenoble
Laboratoire LMC
Grenoble, France
Olivier.Gaudoin@imag.fr

Abstract

The purpose of this paper is to use the framework of hidden Markov chains for the modelling of the failure and debugging process of software, and the prediction of software reliability. The model parameters are estimated using the forward-backward EM algorithm and model selection is done with the BIC criterion. The advantages and drawbacks of this approach with respect to usual modelling are analyzed. Comparison is also done on real software failure data. The main contribution of hidden Markov chain modelling is that it highlights the existence of homogeneous periods in the debugging process, which allow one to identify major corrections or version updates. In terms of reliability predictions, the hidden Markov chain model performs well on average with respect to usual models, especially when the reliability is not regularly growing.

1 Software reliability modelling

Let $X_i, i \geq 1$, be the times between successive failures of a software. After each failure, the software is corrected or not, and restarted. Debugging times are negligible or not taken into account. A huge number of models for the failure process $\mathbf{X} = \{X_i\}_{i\geq 1}$ have been proposed in the last 30 years (see a recent review in Pham, 2000). The most popular class of models, because of its simplicity, is that of Non Homogeneous Poisson Processes (NHPP), for which the failure intensity is a continuous function of time. However, this assumption is not realistic because debugging should induce a discontinuity in failure intensity.

According to Littlewood (1989), two sources of uncertainty exist in the failure behavior of software undergoing debugging. The first source of uncertainty is in the inputs: software inputs are chosen randomly in the input space according to the operational profile. The second source of uncertainty is the effect of debugging. It is logical to assume that the debugging of a fault at a given time depends only of the software state at this time, and not of its past states. This leads to a Markovian modelling of the debugging process. Moreover, software do not wear-out: if a piece of software is not modified, its ability to fail does not change, so the failure intensity between two debuggings should be constant. Then, all these assumptions lead to another class of models, sometimes called the Markov Failure Rate (MFR, Gaudoin 1990) models, for which there exists a Markov process $\mathbf{\Lambda} = \{\Lambda_i\}_{i\geq 1}$ such that, conditionally to $\{\Lambda_i = \lambda_i\}_{i\geq 1}$, the times between failures X_i are independent and exponentially distributed with respective parameters λ_i. Λ_i is the software failure rate after $(i-1)th$ debugging.

All these models assume that there is a correction for each failure, and that the debugging efficiency is homogeneous in time. In practice, after software failures, computers are often rebooted without any correction. Debugging happens when a sufficiently large amount of failures has occurred. When a software is in its operational life, there is a version update or introduction of a new release instead of debugging, but both concepts can be handled in the same way. Even if a correction is done after each failure, most of them are minor and there are sometimes major corrections, which can be considered as equivalent to version updates. Software reliability data generally consist in a list of successive times between failures, and the information on whether a correction has been performed or not, or whether corrections are minor or major, is not available. Thus, it would be interesting to build software reliability models which could take this fact into account. This is the case of the hidden Markov chain (HMC) modelling.

2 Modelling the software failure process with hidden Markov chains

In order to use the framework of hidden Markov chains, it is necessary to assume that the failure rate process $\mathbf{\Lambda}$ takes values in a finite set. Let K be the cardinal number of this set and $\{\lambda^{(1)}, \ldots, \lambda^{(K)}\}$ be the set of possible values for the Λ_i. The assumptions on $\mathbf{X}$ and $\mathbf{\Lambda}$ sum up as follows:

1. $\mathbf{\Lambda}$ is a discrete-valued homogeneous Markov chain starting from $\Lambda_1 = \lambda^{(1)}$; this chain is hidden since the Λ_i are not directly observable;

2. conditional on $\{\Lambda_i = \lambda_i\}_{i \geq 1}$, the times between failures $\{X_i\}_{i \geq 1}$ are independent;

3. conditional on $\{\Lambda_i = \lambda^{(j)}\}$, X_i has an exponential distribution with parameter $\lambda^{(j)}$.

Each trajectory of process $\mathbf{X}$ can be split into homogeneous zones, each zone corresponding to one value $\lambda^{(j)}$ of the failure rate process. In a given zone, the failure rate remains constant. In the software test period, the homogeneous zones can be interpreted as periods where no corrections have occurred after failures, or where the corrections introduced were minor and did not improve significantly the failure rate. The jumps correspond to corrections in the first case and major corrections in the second. In the software operational life, the transitions between zones can be interpreted as introductions of version updates or new releases.

The advantage of the HMC model with regard to NHPP models is that it takes into account the discontinuities in failure intensity caused by the debugging and the no wear-out property of software. The advantage with regard to MFR models is that there is not necessarily a correction after each failure, which leads to the existence of homogeneous periods.

3 Parameter estimation

Let η be the set of all parameters of the hidden Markov chain, given by:

1. The transition probabilities $p_{jl} = P(\Lambda_{i+1} = \lambda^{(l)} | \Lambda_i = \lambda^{(j)})$, $\forall i \geq 1, 1 \leq j \leq K, 1 \leq l \leq K$. The transition parameters amount to the matrix P defined by the p_{jl}.

2. The values $(\lambda^{(1)}, \ldots, \lambda^{(K)})$ of the failure rates.

For any sequence $\{z_i\}_{i \geq 1}$ and any couple of integers (i, j) such that $i < j$, let $\mathbf{z}_i^j$ denote $(z_i, \ldots, z_j)$. The HMC model is a typical case where the complete data can be split into the observed data $\mathbf{x}_1^n$ and the missing data λ_1^n. Then, the estimation of η is done with the EM algorithm (Dempster *et al*, 1977), dedicated to the likelihood maximization in the context of missing values. This iterative algorithm starts from an initial value $\eta^{(0)}$ of the parameters and creates a sequence $\{\eta^{(m)}\}_{m \geq 0}$ whose likelihood grows. The sequence $\{\eta^{(m)}\}_{m \geq 0}$ converges to the consistent solution of the likelihood equations when $\eta^{(0)}$ is close to the optimal solution. Using the results of Baum *et al* (1970), $\eta^{(m+1)}$ is given by:

$$\hat{p}_{jl}^{(m+1)} = \frac{\sum_i E_{\eta^{(m)}}[\mathbb{1}_{\{\Lambda_i = \lambda^{(l)}, \Lambda_{i-1} = \lambda^{(j)}\}} | \mathbf{X}_1^n = \mathbf{x}_1^n]}{\sum_i E_{\eta^{(m)}}[\mathbb{1}_{\{\Lambda_{i-1} = \lambda^{(j)}\}} | \mathbf{X}_1^n = \mathbf{x}_1^n]}$$

$$\hat{\lambda}^{(k),(m+1)} = \left[\frac{\sum_i E_{\eta^{(m)}}[\mathbb{1}_{\{\Lambda_i = \lambda^{(k)}\}} x_i | \mathbf{X}_1^n = \mathbf{x}_1^n]}{\sum_i E_{\eta^{(m)}}[\mathbb{1}_{\{\Lambda_i = \lambda^{(k)}\}} | \mathbf{X}_1^n = \mathbf{x}_1^n]} \right]^{-1}$$

The expectations are computed recursively, using the forward-backward algorithm of Baum *et al* (1970). 3 initial values for $\eta^{(0)}$ are randomly chosen and 50 iterations are run for each value. Thus, three estimators are obtained. The one which has maximal likelihood is used as starting position for the EM algorithm. The algorithm is stopped when 1 000 iterations have been done, or when the relative increase of the log-likelihood is below the threshold $\varepsilon = 10^{-6}$.

4 Restoration of the hidden states

With an estimate of η, it is possible to restore the hidden states, that is to say give a likely value to the unknown state sequence λ_1^n. A natural way to restore the hidden states is to compute their most likely value, conditionally to $\mathbf{x}_1^n$: $\arg\max_{\lambda_1^n} P_\eta(\mathbf{\Lambda}_1^n = \lambda_1^n | \mathbf{X}_1^n = \mathbf{x}_1^n)$. This can be done by the Viterbi algorithm (Forney, 1973). The comparison between original data and restored hidden states is easy since the expected interfailure time when $\Lambda_i = \lambda^{(j)}$ is $1/\lambda^{(j)}$.

The restoration of the hidden states leads to the interpretation of sequences of successive equal states as homogeneous zones in the failure process.

5 Choice of the number of hidden states and transition matrix

In the sections above, the number of possible failure rates K has been considered as known. Clearly, K is not known in practice and has to be estimated. We use the BIC criterion (Bayesian Information Criterion - Kass and Raftery, 1995): $BIC(K) = \log(P_{\hat{\eta}_K}(\mathbf{x}_1^n)) - \frac{\nu_K}{2}\log(n)$, where $\log(P_{\hat{\eta}_K}(\mathbf{x}_1^n))$ is the maximum log-likelihood for the hidden Markov chain with K hidden states and where ν_K is the number of independent parameters in η_K. The BIC criterion is composed of a term measuring the fit between the data and the model and of a term penalizing complex models. Thus, maximizing this criterion hopefully leads to selecting models offering a compromise between fit and complexity.

Until now, no particular assumption has been made on the transition matrix, which means that any improvement or deterioration of the software failure rate is possible. In the case of pure reliability growth models, each debugging reduces the software failure rate. Thus, any transition from a hidden state to a previously visited hidden state must be forbidden. This assumption implies an upper diagonal transition matrix. It is important to take also into account the possibility of imperfect debugging. Then, the return to a previously visited state must be allowed. The easiest way to do so is to allow two transitions for each state (except the initial and final ones): one to the last previously visited state, and one to the next new visited state. This assumption implies a tridiagonal transition matrix. Other types of transition matrix are possible. The choice of a type of matrix can be done by the BIC criterion.

6 Predictive validity and model comparison

The quality of reliability predictions provided by the hidden Markov chain model has to be assessed and compared with those given by other software reliability growth models. The usual method for comparing software reliability predictions is the so-called *U-plot* method (Keiller *et al.*, 1983).

The idea of the method is to compute the $u_i = P_{\hat{\eta}_i}(X_i \leq x_i | \mathbf{X}_1^{i-1} = \mathbf{x}_1^{i-1})$. If the model is appropriate and the estimation is of good quality, then the u_i should be close to a sample of the uniform distribution over $[0,1]$. This closeness or predictive validity is measured by the Kolmogorov-Smirnov distance KS between the empirical CDF of the u_i and the true uniform CDF, which is $F(x) = x$ on $[0,1]$. The *U-plot* is the plot of the empirical CDF of the u_i. If several models are competing, the "best model" is the one for which KS is the smallest.

7 Application and conclusion

The hidden Markov chain model has been used on two groups of real software failure data sets. The first group consists of times between failures of nine American control-command software in the test period and operational life (Musa, 1979). The second group consists of times between failures of four complex French software in the test period (Gaudoin, 1990). Time unit is running clock time in both cases.

First of all, we present a complete treatment of one of these data sets, M40, for which $n = 101$. Figure 1 shows that the BIC criterion is maximum for a model with three hidden states and an upper diagonal transition probability matrix. For this model, the parameter estimates are:

$$\begin{bmatrix} \hat{\lambda}^{(1)} & \hat{\lambda}^{(2)} & \hat{\lambda}^{(3)} \end{bmatrix} = 10^{-4} * \begin{bmatrix} 0.5035 & 0.0908 & 0.0175 \end{bmatrix}$$

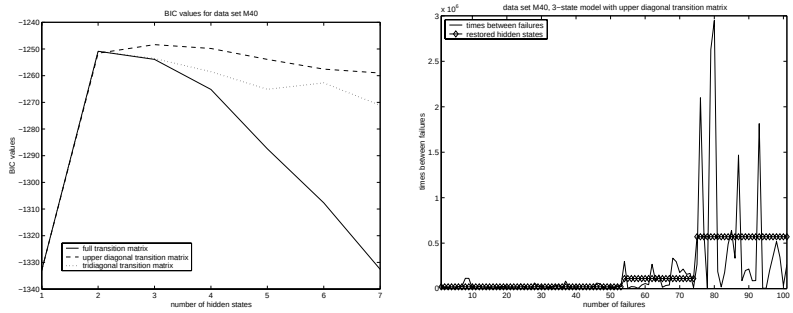

Figure 1: Model selection with BIC and restoration of the hidden states for data set M40

$$\hat{P} = \begin{bmatrix} 0.9809 & 0.0191 & 0 \\ 0 & 0.9502 & 0.0498 \\ 0 & 0 & 1.0000 \end{bmatrix}$$

Then, it is easy to restore the hidden states, as shown in figure 1. Three homogeneous periods clearly appear in this figure.

Finally, the HMC model is compared with some of the most usual software reliability models, using the U-plot method. The main results of this study are the following ones.

- For all data sets, the selected number of hidden states is very small, from 1 to 4. This means that for these software, very few major corrections seem to have occurred during the debugging process.
- Logically, the upper diagonal transition matrix gives the best result for data sets exhibiting almost pure reliability growth. For data sets for which reliability is sometimes decreasing (probably due to imperfect debugging), full or tridiagonal matrices are better.
- In terms of predictive validity, usual models perform better than HMC when there is a regular reliability growth, and HMC is the best when there are some significant imperfect debuggings.
- Long homogeneous periods are detected by the restoration of hidden states for almost all data sets, leading to a clear interpretation in terms of identification of major corrections.

References

Baum, L. E., Petrie, T., Soules, G. and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Ann. Math. Stat. 41*, 164–171.

Dempster, A., Laird, N. and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm (with discussion). *J.R.S.S. B 39*, 1–38.

Forney Jr., G. D. (1973). The Viterbi Algorithm. *Proceedings of the IEEE 61*, 268–278.

Gaudoin, O. (1990). Outils statistiques pour l'évaluation de la fiabilité des logiciels. PhD thesis, Joseph Fourier University, Grenoble (in French).

Kass, R. E. and Raftery, A. E. (1995). Bayes factors. *J.A.S.A. 90*, 773–795.

Keiller, P. A., Littlewood, B., Miller, D. R. and Sofer, A. (1983). Comparison of software reliability predictions. In *Proc. 13th IEEE Int. Symp. on Fault Tolerant Computing*, pp. 128–134. Milano: IEEE Computer Society Press.

Littlewood, B. (1989). Predicting software reliability. *Phil. Trans. Royal Stat. Society, Series A 327*, 513–527.

Musa, J. D. (1979) Software reliability data. *Technical Report*, Rome Air Development Center.

Pham, H. (2000). *Software reliability*. Springer.